Build your Excel skills with these 10 power tips

By Susan Harkins

TechRepublic

Contents

3

How to add a dropdown list to a cell

You can create a simple list control for your users in a sheet. Doing so reduces errors by limiting input from the keyboard and facilitates input tasks for your users. Adding the control to an Excel sheet isn't intuitive; the option is in the Data Validation feature. Fortunately, once you know the feature exists, it's easy to implement.

You need only two things: a list and a data entry cell. **Figure A** shows a simple dropdown list in an Excel sheet. Users click the dropdown arrow to display a list of items stored in A1:A4. If a user tries to enter something that isn't in the list, Excel rejects the entry (by default).
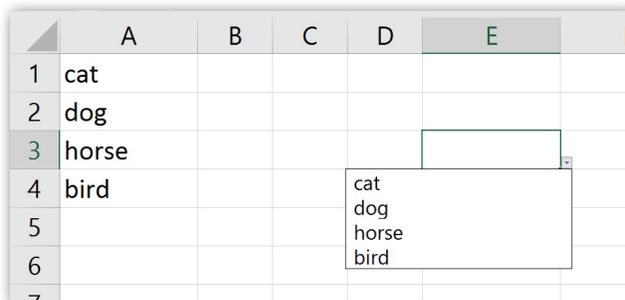

Figure A: You can use this list to populate a list control for easy data entry.

To add this dropdown list to a sheet, do the following:

1. Create the list in cells A1:A4.
2. Select cell E3, the data entry cell.
3. Click the Data tab.
4. Click the Data Validation option in the Data Tools group.
5. Choose List (**Figure B**) from the Allow option's dropdown list.
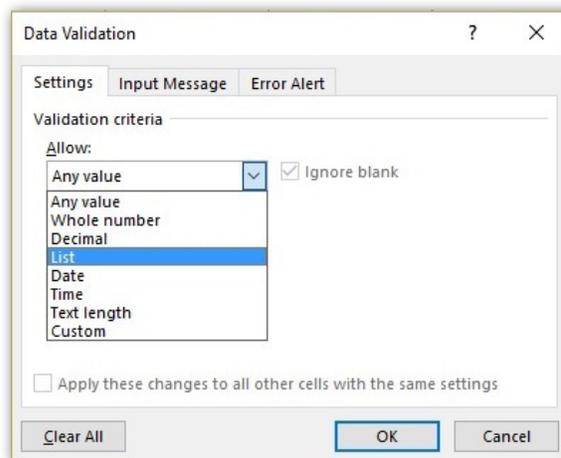

Figure B: Choose the list option.

6. Click the Source control and select cells A1:A4 (**Figure C**). Alternately, simply enter the reference *=$A$1:$A$4*. Make sure the In-Cell Dropdown option is checked. If you uncheck this option, Excel still forces users to enter only list values, but it won't present a dropdown list.
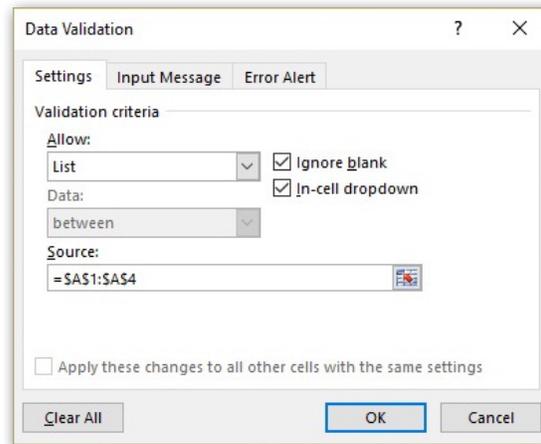
7. Click OK.

Figure C: Identify the list items.

Users click the dropdown and choose an item to enter a value into cell E3, and E3 equals the selected value. It couldn't be simpler!

Notes

It's worth pointing out that the dropdown arrow is visible only when the cell is active. To make the control easier to use, consider applying a fill color or border to the cell. If you believe users will need more information, explore the Input Message and Error Alert options. In addition, if you use this feature in a Table, Excel will automatically copy the control to new rows.

5

Build a dynamic chart using a Table object

If you want to advance beyond basic spreadsheet skills, creating dynamic charts is a good place to begin that journey. The key is to define the chart's source data as a dynamic range. By doing so, charts based on that data will automatically reflect changes. Fortunately, this is easy to accomplish if you're willing to use the Table feature (available in Excel 2007 and later).

First, convert a data range into a Table as follows:

1. Select the data range. Using the data set shown in **Figure A**, you'd select A1:E3.

Figure A: Convert the data range into a Table.

2. Click the Insert tab.

3. In the Tables group, click Table.

4. If the Table doesn't have headers, be sure to uncheck the My Table Has Headers option.

5. Click OK, and Excel will format the data range as a Table.

Any chart you build on the Table will update automatically. To illustrate this efficient behavior, create a quick column chart as follows:

1. Select the Table.

2. Click the Insert tab.

3. In the Charts group, choose the first 2-D column chart in the Chart dropdown to create the simple column chart shown in **Figure B**.

Figure B: Base a column chart on the data.
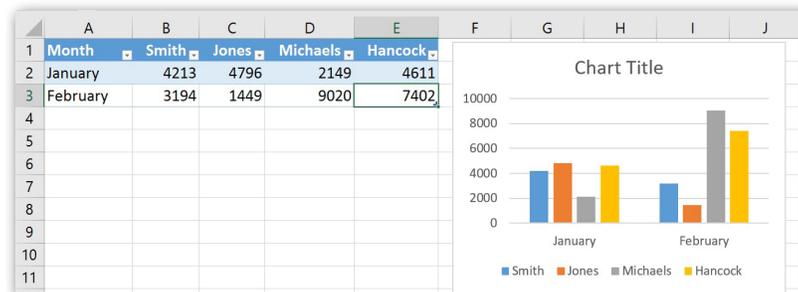
Copyright ©2016   CBS Interactive Inc. All rights reserved.

Now, update the resulting chart by adding values for March, as shown in **Figure C** and watch the chart update automatically.
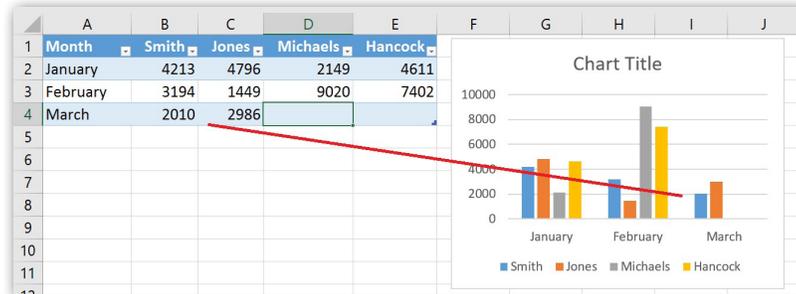

Figure C: The chart updates automatically.

Count the number of records that fall between two dates

Many records include a date stamp of some sort. Usually the date marks an event or the record's input date. Fortunately, counting the number of records that fall between two dates is easy using the COUNTIFS() function. We can illustrate this technique using the simple sheet shown in **Figure A**.

The formula in cell B3:

`=COUNTIFS(A5:A12,">="&B1,A5:A12,"<="&B2)`

returns the number of dates (records) in A5:A12 that fall between the dates entered in B1 and B2.

Figure A: We'll count records between date values in the Date field.

The COUNTIFS() function uses the following syntax to specify multiple criteria to determine which values in a range to count:

`COUNTIFS(countrange1, criteria1, [countrange2, criteria2]...)`

In this example, the count range is the same for each criteria set, the dates in column A. The first criteria set counts all the dates in A5:A12 that are equal to or fall *after* the date value in B1. The second criteria set counts dates that are equal to or occur *before* the date value in B2.

The only stipulation with this particular setup is that the value in B1 must be less than the value in B2. If you enter them backward, the function will return 0 instead of an error. To apply a conditional format to alert your users, do the following:

1. Select B2.

2. Click the Home tab and then click Conditional Formatting in the Styles group.

3. Choose New Rule.

4. Choose the Format Only Cells That Contain option.

5. Choose Less Than from the second dropdown.

6. Enter =*$B$1* in the expression control.

7. Click Format.

8. Click the Fill tab, choose a background color, and click OK. **Figure B** shows the rule, expression, and format.
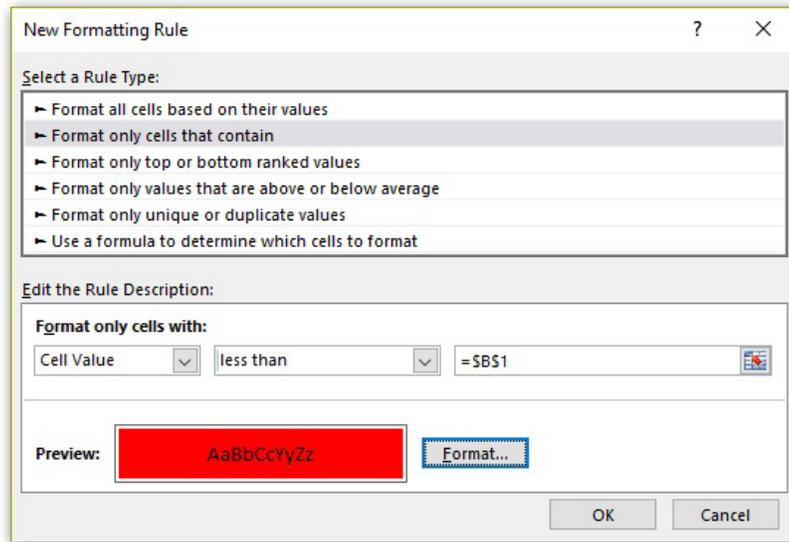
9. Click OK to return to the sheet.

Figure B: Use a fill color to highlight an input error.

The format won't stop a user from entering an incorrect date, but it will alert them that the value isn't appropriate, as shown in **Figure C**.

Figure C: A conditional format warns users when the date in B2 is less than the date in B1.

How to sum values in a filtered list

Filters offer a powerful and easy-to-use way to limit data to specific records. To sum a filtered set, you can try a SUM() function, but you might get a surprise—well, I can promise you'll get a surprise. **Figure A** shows a filtered list. Specifically, the filtered list displays only those records where the Unit Price value is greater than or equal to 50.

Figure A: You can tell by the row numbers to the left that many rows are hidden.

Our technique uses the Table object's filtering behaviors. If you don't want to use a Table, you can filter records using Excel's built-in filtering feature on the Data tab in the Sort & Filter group.

Before we look at SUM()'s limitations when working with filtered lists and how to avoid those limitations, let's quickly review the data. First, the data is formatted as a Table object (available in Excel 2007 and later). To convert a data range to a Table, do the following:

1. Click anywhere inside the data range.
2. On the Insert tab, click Table in the Tables group.
3. In the resulting dialog, uncheck or check the My Table Has Headers option (**Figure B**).
4. Click OK.

Figure B: Convert a data range into a Table.

To achieve the result shown in Figure A, apply the following filter:

1. From the Units On Order dropdown, choose Number Filters and then select the Greater Than Or Equal To option (**Figure C**).
2. In the resulting dialog, enter *50* (**Figure D**) and click OK.
3. In cell E47, enter a SUM() function for E3: E33 (if you drag) and E3:E46 if you enter the references manually.

You can tell that the result isn't correct; the value is too high—but why? The SUM() function is evaluating all the values, not just the filtered values. There's no way for the SUM() function to know that you want to exclude the filtered values in the referenced range.

10
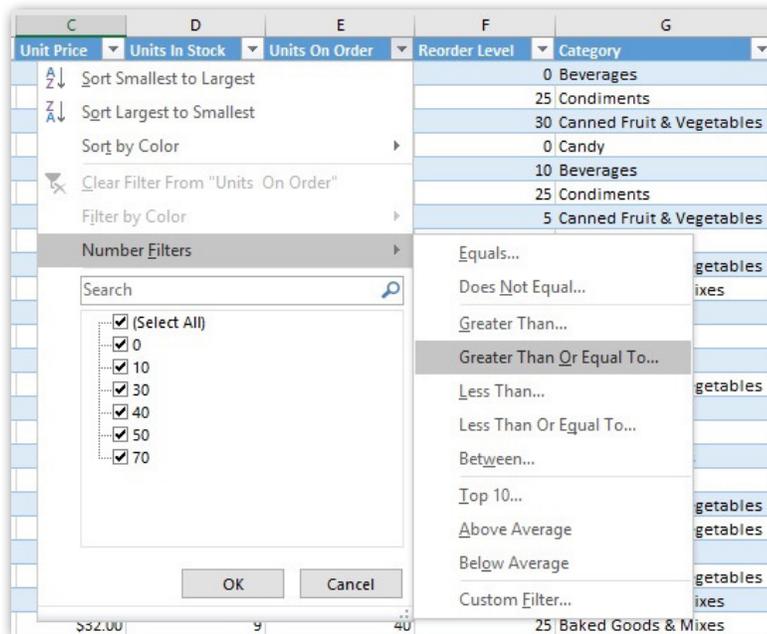
Figure C: Choose a filter.

Fortunately, there's more than one solution. First, click AutoSum in the Editing group on the Home tab. As you can see in **Figure E**, Excel automatically enters a SUBTOTAL() function instead of a SUM() function. This function references the entire list of values, but it evaluates only the filtered values.
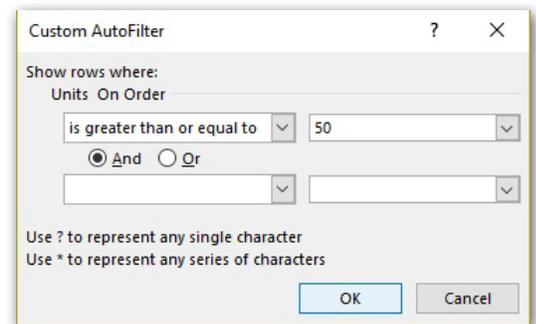
Figure D: Enter the criteria.
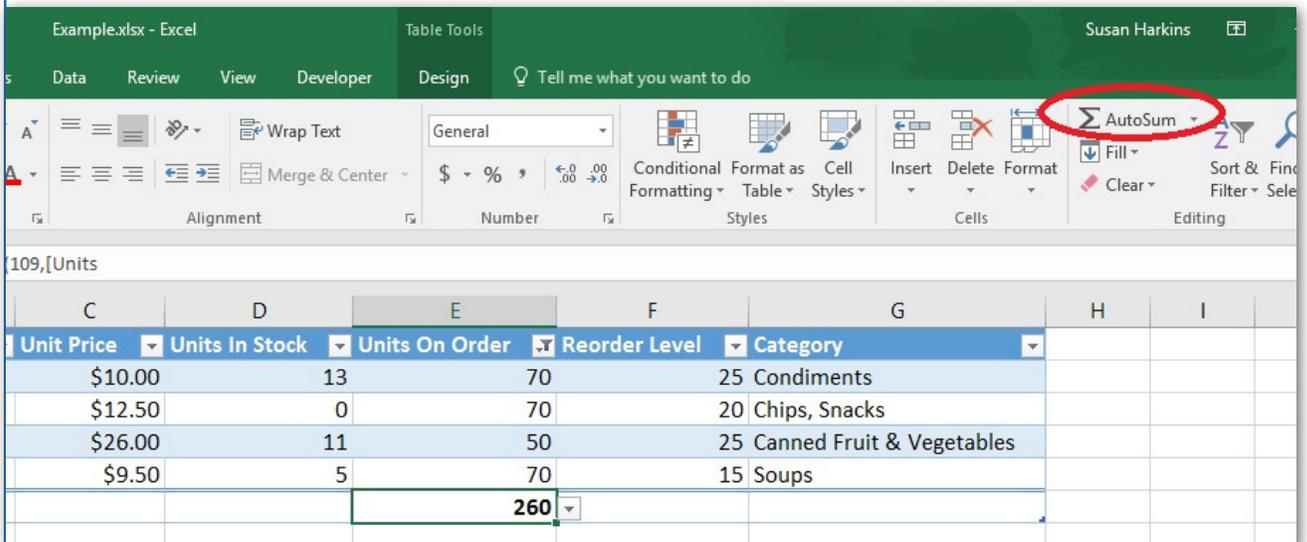
Figure E: SUBTOTAL() knows to evaluate only the visible values in a filtered set.

There's a second way that's a bit cleaner: Use the Table's Total Row. Before you do, delete the SUBTOTAL() function in E47 and clear the filter. Adding the Total Row is simple:

1. Click anywhere inside the Table.

2. Click the contextual Design tab.

3. In the Table Style Options group, check Total Row (**Figure F**) and Excel will expand the Table. Look for the thin blue line surrounding the new row.
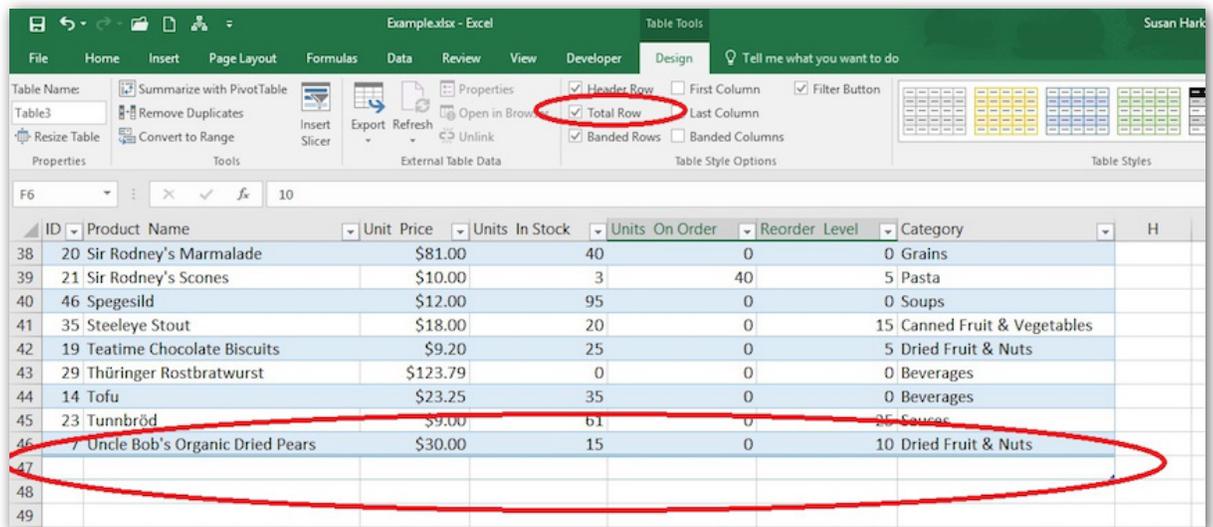
Figure F: Display the Total Row.

4. Choose Sum from the Units On Order dropdown (**Figure G**). Each cell in the Total Row offers this drop-down. Evaluating the entire data set, the function returns 420.

Figure G: Select a function.

5. Apply the filter you used earlier (>=50) and the function automatically updates, displaying the same results you saw in Figure E. If you check the Formula bar, you'll see that Excel entered SUBTOTAL(), not SUM(). Excel is smart enough to anticipate your needs.

Hide everything but the working area in an Excel worksheet

You usually hide a column or row to conceal or protect data and formulas. But you can also hide unused regions of a sheet to keep users from exploiting those areas or to help them stay on task by preventing them from wandering. By inhibiting access to unused rows and columns, you present a sheet that focuses on just the work area. To demonstrate, we'll use the sample worksheet shown in **Figure A**, which has a small working area and a lot of unused rows and columns.

Figure A: Let's inhibit access to unused areas of this sheet.

Before you hide anything, make sure you don't inadvertently hide an obscure work area. To find the last cell in the sheet's used range, press Ctrl + End. If you find anything, deal with it before hiding that area.

When ready, hide unused rows as follows:

1. Select the row beneath the sheet's last used row. (Select the row header to select the entire row.) In our example sheet, select row 9.

2. Press Ctrl+Shift+Down Arrow to select every row between the selected row and the bottom of the sheet.

3. On the Home tab, click Format in the Cells group.

4. From the dropdown, choose Hide & Unhide and then choose Hide Rows (**Figure B**).
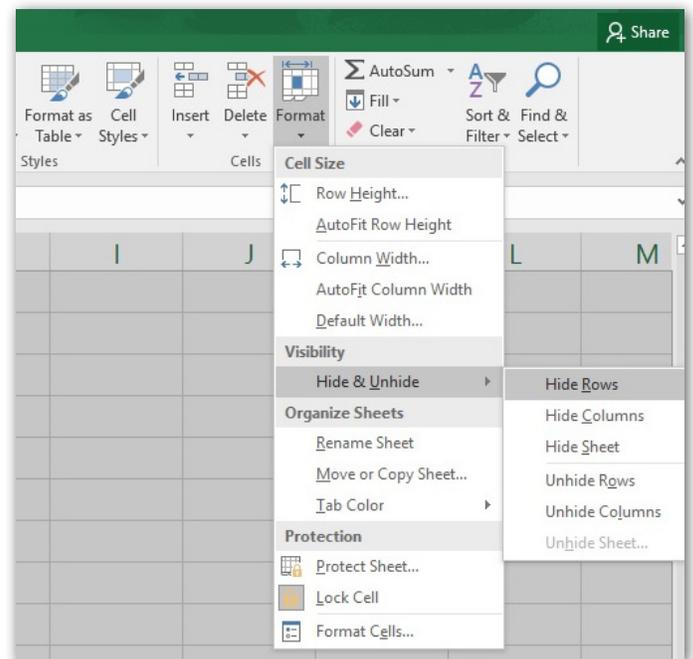
Figure B: Choose the Hide format.

5.   Repeat this process to hide unused columns, only select the column header in the first empty column.

6.   Press Ctrl+Shift+Right Arrow.

7.   Choose Hide Columns from the Format dropdown's Hide & Unhide option.

Figure C shows the resulting sheet. The only cells users can select are those visible in the sheet.

Figure C: Hide unused rows and columns.

To unhide rows and columns, select the entire sheet. Then, do the following;

1.   Click the Home tab.

2.   Choose Hide & Unhide from the Format dropdown.

3.   Choose Unhide Rows and Unhide Columns appropriately.

To complete the picture, you might also want to turn off the gridlines. To do so, uncheck Gridlines in the Show/Hide group on the View tab. With the extraneous rows, columns, and gridlines hidden, you present a cleaner, less busy interface for your Excel users. In short, users have fewer options to distract them.

Sometimes user tampering causes trouble and you're the lucky one who has to make things right again. But seeing only the actual spreadsheet cells tells users this isn't an ordinary workbook. Instead of seeing a canvas they can alter, they see a custom application. They do their work, and they move on. Be aware, however, that a user who knows how to unhide the rows and columns can quickly get around this tactic. Don't apply it as a true security feature because it isn't.

Using Excel's FIND() and MID() functions to extract a substring when you don't know the start position

Excel's string functions provide a way for returning characters and subsets from an existing string. For instance, using LEFT(), you can quickly return the first *n* characters in a string. Similarly, with RIGHT(), you can return the rightmost *n* characters. MID() returns *n* characters from a specific starting position. The functions rely on consistency. When that's missing, consider combining FIND(). To demonstrate, I'll show you how to combine FIND() and MID() to return characters from the middle of the string when the start point is inconsistent.

About the functions

Before we start building parsing expressions, let's quickly review the functions we'll use: MID() and FIND(). Excel's MID() function requires three arguments and uses the following syntax:

`MID(text, start, num)`

where *text* identifies the whole string, *start* denotes the position of the first character to return, and *num* specifies the number of characters to return. In other words, MID() returns characters from a string by specifying the first character's position and the last character by its relationship to the first character.

Excel's FIND() function has two required arguments and uses the following syntax:

`FIND(findtext, withintext, [start])`

where *findtext* is the substring you want to find, *withintext* is the text you're searching, and the optional *start* argument specifies the character at which to start the search. If omitted, *start* is 1, the first character in *withintext*.

Use FIND() to generate MID()'s start point

Knowing that the substring always begins with a certain position makes it easy to use the MID() function. For example, MID(*text*, 4, 3) will always return three characters starting with the fourth character in *text*. But what do you do when the substring could start anywhere? As long as there's something in the string that identifies the start of the substring—an anchor—you can combine FIND() and MID() to accomplish the mission. If the MID() function's *start* position is unknown or inconsistent, use FIND().

To illustrate this technique, we'll use a simple expression to return the first two characters following the hyphen in the variable-length strings

	A	B
1	**SOURCE**	
2	K2445-B2100	
3	K-B3150	
4	K24112-B3175	
5	K252-B4100	
6	K25001-B4250	
7	K266221-B4250	
8	K26-B4250	

Figure A: MID() alone can't return the first two characters following the hyphen character for each string.

shown in **Figure A**. It's impossible using only MID() because the hyphen's position is inconsistent.

Here's the trick: Use FIND() to return the position of the hyphen in the string and then use that result as the MID() function's *start* argument. **Figure B** shows a simple FIND() function at work.

To achieve this result, enter the function:

`=FIND("-",A2)`

into cell B2 and copy it to B3:B8. To extract the first two characters that *follow* the hyphen, we add 1 to FIND()'s result. So the expression:

`=Mid(A2,Find("-",A2)+1,2)`

returns the first two characters following the first hyphen, as shown in **Figure C**.
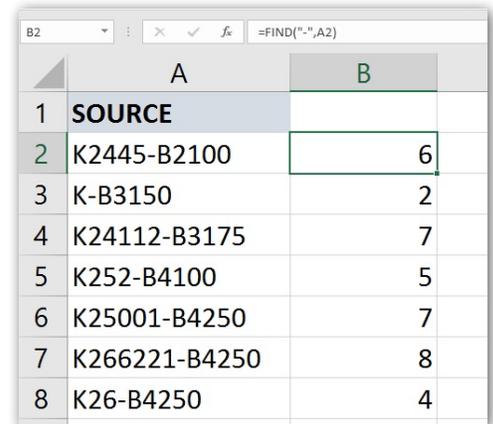
	A	B
1	SOURCE	
2	K2445-B2100	6
3	K-B3150	2
4	K24112-B3175	7
5	K252-B4100	5
6	K25001-B4250	7
7	K266221-B4250	8
8	K26-B4250	4

fx: `=FIND("-",A2)` (B2)

Figure B: FIND() returns the position of the hyphen in each of the strings in column A.
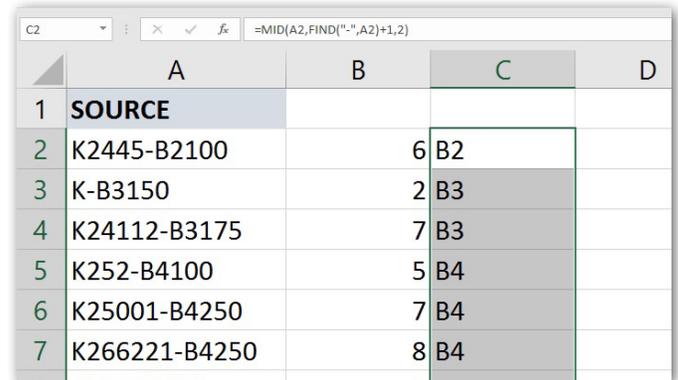
	A	B	C	D
1	SOURCE			
2	K2445-B2100	6	B2	
3	K-B3150	2	B3	
4	K24112-B3175	7	B3	
5	K252-B4100	5	B4	
6	K25001-B4250	7	B4	
7	K266221-B4250	8	B4	

fx: `=MID(A2,FIND("-",A2)+1,2)` (C2)

Figure C: MID() uses the result of a FIND() function as its start argument.

How to suppress 0 values in a chart

Charting 0s isn't wrong, but you won't always want to display 0 values in charts. Your data and the chart's purpose will guide you in this decision. When you don't want to display these values, you have a few choices, and some work better than others. We'll review a few that offer quick but limited results with minimum effort.

About the example data

Figure A shows the data and initial charts we'll use throughout this article. Right now, the charts display 0 values. These techniques are autonomous; if you use the same data set, be sure to undo each solution before you start the next. Simply close the file and reopen without saving. (If you're using an earlier version, your results might vary.)
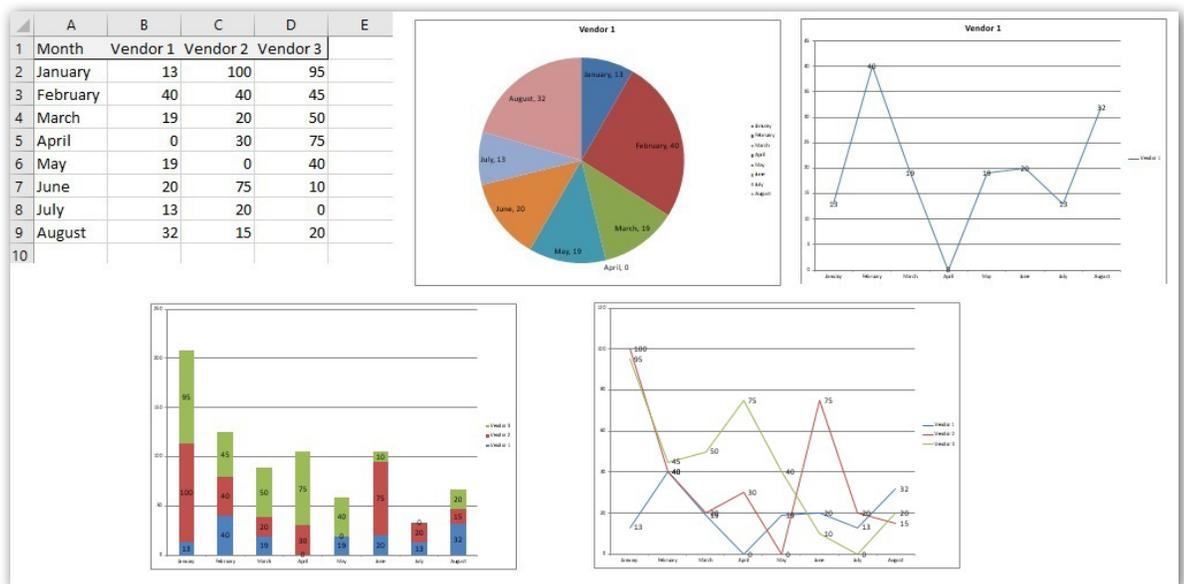
Figure A: We'll use this sample data set and these charts.

The pie and single line charts reflect the data in column B for Vendor 1. The other two charts have three series: Vendor 1, Vendor 2, and Vendor 3. This setup simplifies all the examples. Now that you're familiar with the example data, let's review a few methods for suppressing the 0 values in our example charts.

Easiest but limited

You might try removing 0 values altogether if it's a literal 0 and not the result of a formula. Unfortunately, this simplest approach doesn't always work as expected. The stacked bar responds well to this solution. The pie chart doesn't chart the missing 0s, but the legend displays the category label. Neither line chart handles the missing 0s well. If you removed the 0 values in the sheet, reenter them before you continue.

Hiding the 0s

You can try hiding the 0s; there are two ways. First, try disabling the Show A Zero In Cells That Have Zero Value option as follows:

1. Make sure the sheet with the charted data is active.

2. Click the File tab and choose Options.

3. Choose Advanced in the left pane.

4. In the Display Options For This Worksheet section, choose the appropriate sheet from the dropdown menu.

5. Uncheck the Show A Zero In Cells That Have Zero Value option (**Figure B**).

6. Click OK. The 0 values still exist, and you can see them in the Formula bar, but Excel won't display them.
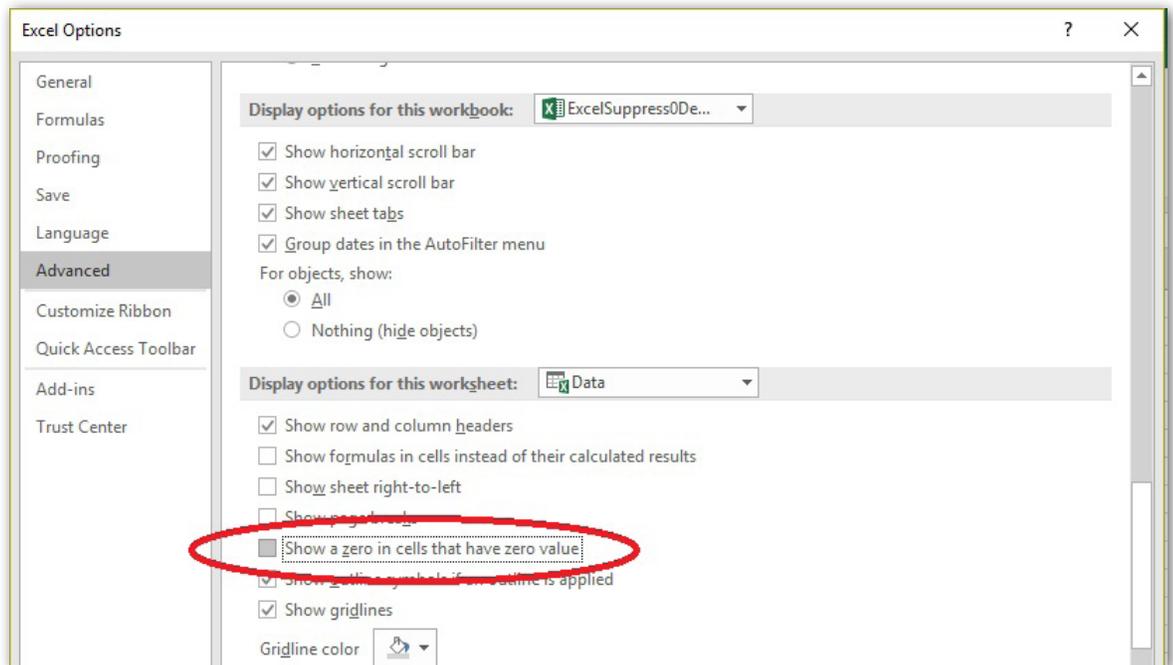
Figure B: Uncheck this option.

This method has little to no impact; the chart treats the 0 values as if they're still there, because they are. Be sure to reset the option before you continue.

You might also try using the following format that hides 0s:

1. Select the data range.

2. Click the Number group's dialog launcher (Home tab).

3. In the resulting dialog box, choose Custom from the Category list.

4. In the Type control, enter *0,0;;;* (**Figure C**).
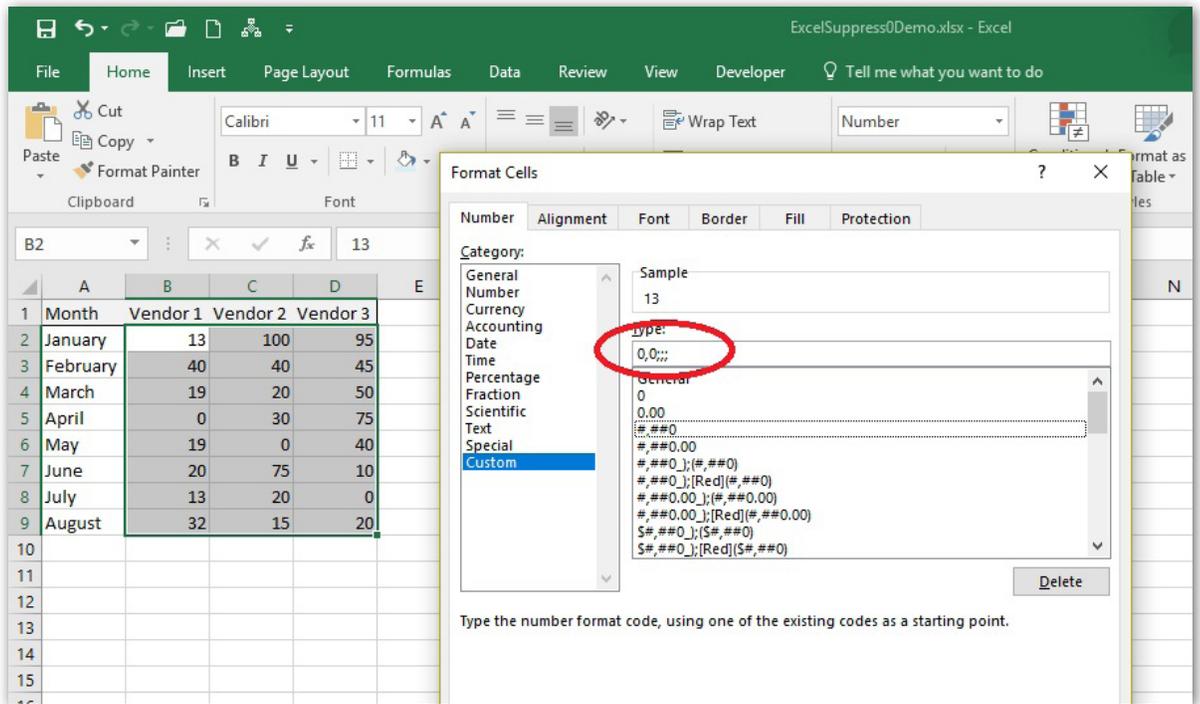
Figure C: Enter the custom format code.

5. Click OK. The stacked bar, shown in **Figure D**, is the only winner.
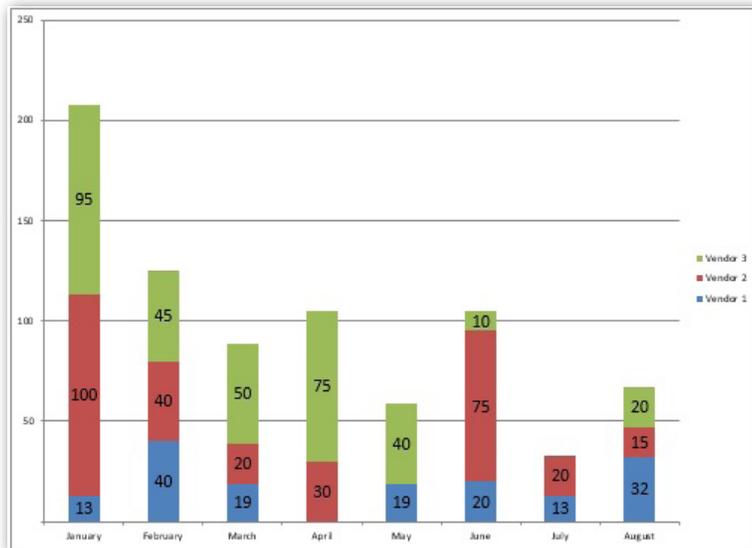
Figure D: This technique works for the stacked bar chart.

These techniques are easy to apply. It won't hurt to try them first, but don't expect a panacea for every chart. Remove the custom format before you continue.

Charting a filtered data set

If you have a single data series, you can filter out the 0 values and chart the results. Like the methods discussed above, it's a limited choice. Use it when it works—just know that it won't always work. We can illustrate this technique by adding a filter to the Vendor 1 column:

1. Click inside the data range.

2. On the Data tab, click Filter in the Sort & Filter group.

3. Click Vendor 1's dropdown and uncheck 0 (**Figure E**).

4. Click OK to filter the column.
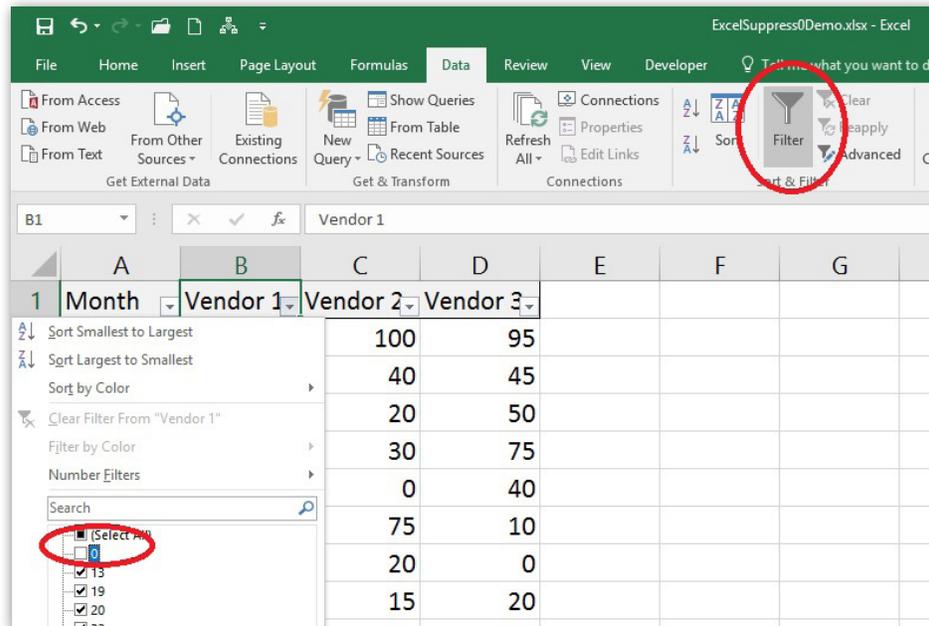
**Figure E: Uncheck the 0 value option.**

Figure F shows the two single series charts based on the filtered data in column B. Neither displays the 0 value or the category label. However, the chart updates, displaying the 0 values, when you remove the filter. On the other hand, if your chart is a one-time task, filtering offers a quick fix. Clear the filter before you continue.

Replace 0s with NA()

Perhaps the most permanent fix is to replace literal 0 values with the NA() function using Excel's Find And Replace feature. Excel won't chart #N/A values. If the data is updated regularly, you might even enter NA() for 0s from the get-go, which will eliminate the problem altogether. However, that's not always practical.
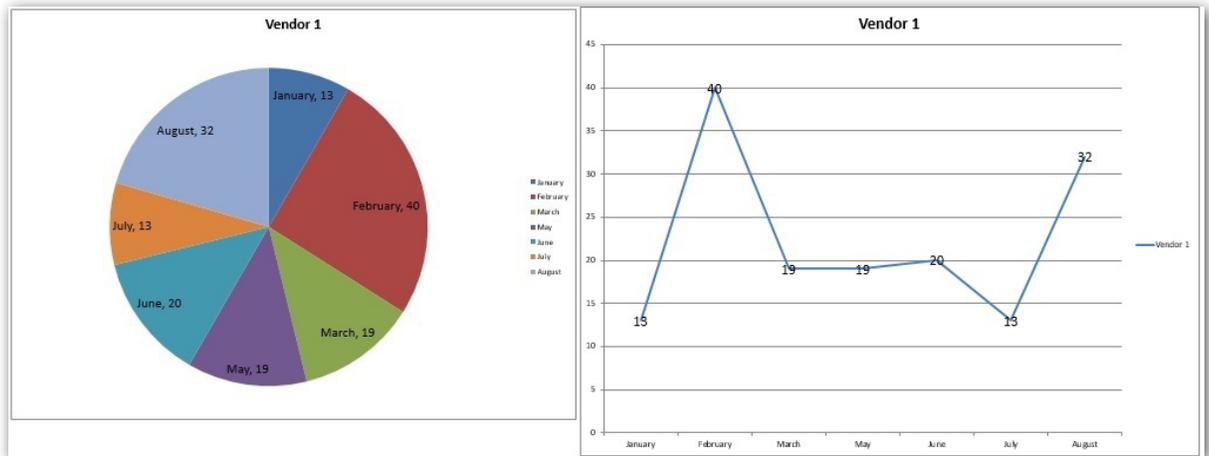
Figure F: The filtered set contains no 0 values in the Vendor 1 column.

First, use Excel's Replace feature to replace the 0 values in the example data set with the NA() function:

1. Select the data set (in this case, it's B2:D9).

2. In the Editing group (on the Home tab), click the Sort & Filter option and choose Replace or simply press Ctrl+H.

3. Enter *0* in the Find What control.

4. Enter *=NA()* in the Replace control.

5. If necessary, click Options to display additional settings.

6. Check the Match Entire Cell Contents option (**Figure G**).
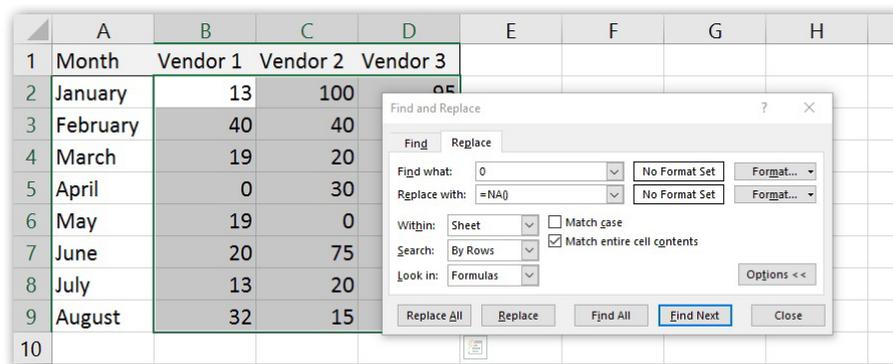
Figure G: Check for 0 values.

7. Click Replace All, and Excel will replace the 0 values.

8. Click OK to dismiss the confirmation message and then click Close.

None of the charts display the #N/A values, but the single series charts still display April in the axis and the legend, which might not matter. If you're working with the results of formulas that might return 0 instead of literal values, you can use an IF() function to return the #N/A error using the following syntax:

```
=IF(formula=0,NA(),formula)
```

No panacea

There isn't an easy one-size-fits-all solution for the challenge of creating 0-less charts. If you're displaying 0s for reporting purposes, you might need to maintain two data sets: one for reporting and one for charting. Removing the category label from the axis dynamically for charting is even harder. On the other hand, if your requirements aren't quite so strict, one of the solutions discussed here should be adequate.

22

10 ways to reference workbooks and sheets using VBA

Referencing workbooks and sheets programmatically generates a lot of confusion because there are so many possibilities. No method is superior; they all have their place. The purpose at hand will define which referencing method is the most efficient. (You must save your workbook as a macro-enabled workbook, .xlsm, to execute code.)

1: Reference the active workbook

VBA's ActiveWorkbook property refers to the workbook with the focus. It's perfectly acceptable to use this property to reference the active workbook from code inside the active workbook. However, it's invaluable when referencing the active workbook remotely. For example, after passing data to an active workbook, you'd probably want to save that workbook, which is a simple task for the ActiveWorkbook property. The following procedures use the ActiveWorkbook property to close the active workbook:

```
Sub CloseActiveWBNoSave()

  'Close the active workbook without saving.

  ActiveWorkbook.Close False

End Sub

Sub CloseActiveWBWithSave()

  'Close the active workbook and save.

  ActiveWorkbook.Close True

End Sub

Sub CloseActiveWB()

  'Close the active workbook; let user choose whether to save.

  ActiveWorkbook.Close

End Sub
```

This save doesn't require the workbook's name, path, and so on. If you need such information, use ActiveWorkbook's Path and Name properties as follows:

```
Function GetActiveWB() As String

  GetActiveWB = ActiveWorkbook.Path & "\" & ActiveWorkbook.Name

End Function
```

2: Reference the workbook that's running code

ActiveWorkbook evaluates the workbook with the focus; ThisWorkbook refers to the workbook that's running the code. The following procedure will return the name of the workbook running the code, which may or may not be the active workbook:

```
Function GetThisWB() As String

  GetThisWB = ThisWorkbook.Path & "\" & ThisWorkbook.Name

End Function
```

3: Reference workbooks in the Workbooks collection

The Workbooks collection contains all the *open* Workbook objects. Using the Workbooks property, you can refer to open workbooks. For instance, the following procedure populates a list box in a user form with the names of all open workbooks:

```
Private Sub UserForm_Activate()

  'Populate list box with names of open workbooks.

  Dim wb As Workbook

  For Each wb In Workbooks

    ListBox1.AddItem wb.Name

  Next wb

End Sub
```

Closing all the open workbooks is a bit easier, as the following procedure shows:

```
Sub CloseAllWB()

  'Close all open workbooks.

  Workbooks.Close

End Sub
```

4: Explicitly reference a workbook

If you know the name of the open workbook you want to reference, an explicit reference might be the most efficient method. Although an explicit reference is easy, it does require a stable situation. For example, the following procedure activates an open workbook, as determined by the passed string in wbname:

```
Function ActivateWB(wbname As String)

  'Activate wbname; wbname must be open.

  Workbooks(wbname).Activate

End Function
```

Pass the name of the workbook you want to activate as follows (be sure to include the extension):

```
ActivateWB("HumanResources.xlsx")
```

The following function also uses the Workbooks property to determine whether a specific workbook is currently open:

```
Function IsWBOpen(wbname As String) As Boolean

  'Determine if workbook is open.

  Dim wb As Workbook

  On Error Resume Next

  Set wb = Workbooks(wbname)

  IsWBOpen = Not wb Is Nothing

End Function
```

If wbname is open, the function returns True. When not open, the function returns False.

5: Reference workbooks by index

Perhaps the least stable method for referencing a workbook is to use its index value. Excel assigns index values to workbooks as you open them. The first workbook opened has an index value of 1, the second workbook opened has an index value of 2, and so on.

Index values pose a special problem because they change when you delete a Workbook object from the collection; index values slip down a notch, accordingly. For example, suppose you have three open workbooks with the following index values:

- HumanResources.xlsx – 1
- 0908002.xlsx - 2
- ExcelStatisticalFunctions - 3

If a particular task depends on all three workbooks always being open, using the index values can generate mistakes. For instance, the statement:

```
Workbooks(1).Activate
```

activates HumanResources.xlsx as long as it's open. If you close HumanResources.xlsx, the index values for ExcelStatisticalFunctions and 0908002.xlsx change: ExcelStatisticalFunctions becomes 2 and 0908002.xlsx becomes 1. As a result, the above statement activates 0908002.xlsx, not HumanResources. That may or may not be what you want. Using index values to reference workbooks isn't wrong, but you must understand the inherent behaviors to avoid errors that can be difficult to troubleshoot.

6: Reference the active sheet

If you don't specify an object qualifier, the ActiveSheet property defaults to the active sheet in the active workbook. For instance, to retrieve the name of the active sheet, you'd use a function similar to the following:

```
Function GetActiveSheet() As String

  GetActiveSheet = ActiveSheet.Name

End Function
```

This property is read-only; you can't use it to activate a sheet.

7: Reference Worksheet objects

The Worksheets collection contains all the open sheet objects in a workbook. Using a simple For Each loop, you can cycle through the collection. For example, the following code populates a list box control with the names of all the sheets in the active workbook:

```
Private Sub UserForm_Activate()

  'Populate list box with names of sheets

  'in active workbook.

  Dim ws As Worksheet
```

```
   For Each ws In Worksheets

      ws.Select

      ListBox1.AddItem ws.Name

   Next ws

End Sub
```

The Sheets and Worksheets collections both contain Worksheet objects, but the Sheets collection contains both worksheets and chart sheets.

8: Explicitly reference sheets

Use the Worksheets property to explicitly reference a sheet. For instance, use this type of reference to delete a specific sheet as follows:

```
Function DeleteSheet(shtname As String)

   'Delete shtname.

   Application.DisplayAlerts = False

   Worksheets(shtname).Delete

   Application.DisplayAlerts = True

End Function
```

9: Reference sheets by index

Index values come in handy when you don't care about specific sheets, but only their number or order. Granted, that's not going to be a common task, but occasionally, referencing by index values can come in handy. The following procedure adds and deletes sheets based on the number of sheets you want:

```
Function ControlSheetNumber(intSheets As Integer)

   'Add or delete sheets to equal intSheets.

   Application.DisplayAlerts = False

   'Delete sheets if necessary

   While Worksheets.Count > intSheets

      Worksheets(1).Delete
```

```
   Wend

   'Add sheets if necessary

   While Worksheets.Count < intSheets

      Worksheets.Add

   Wend

   Application.DisplayAlerts = True

End Function
```

Use caution when executing this function in an existing workbook because it deletes the first Sheet object in the collection, even if that sheet contains content. This function is most useful when creating new workbooks programmatically.

10: Refer to a sheet's code name property

Code that refers to a Worksheet object by the name on the sheet's tab runs the risk of generating an error if you change the sheet's name. One way to safeguard code is to use the sheet's code name property in your code. This property setting is the sheet's default name, which Excel assigns when you create it: Sheet1, Sheet2, and so on. Changing the sheet's name (displayed by the sheet tab) won't change the sheet's code name.

You must use the Visual Basic Editor (VBE) to view and change this property; you can't change it programmatically. There are two similar properties, so don't confuse them. The (Name) property at the top of the list in the properties window (**Figure A**) is the code name. Code names must start with a letter character. The Name property (without parentheses) toward the bottom of the list is the name Excel displays on the sheet tab.
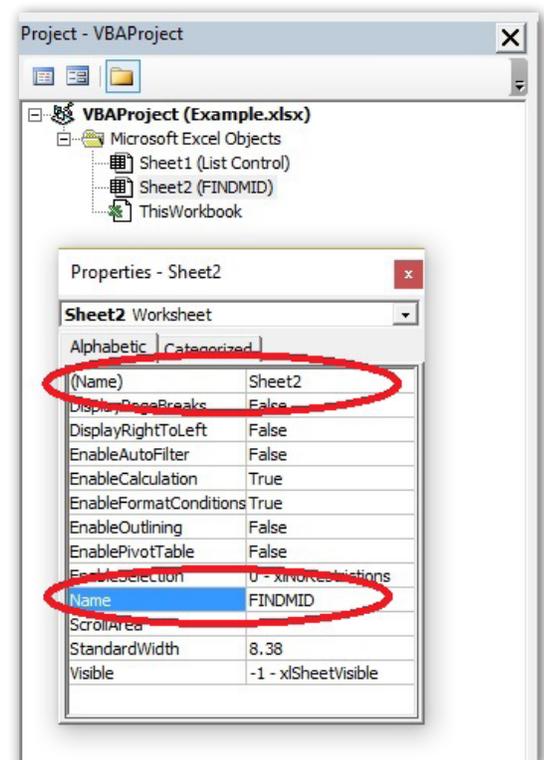

Figure A: Access the sheet's code name using the VBE.

Use a custom format to display easier-to-read millions

Large numbers can be difficult to read, especially if there are many of them. To improve readability, you might want to reduce the number of digits, without losing the number's scale. For example, the sheet shown in **Figure A** shows a lot of large numbers, some in the millions. We can reduce the number of digits using a custom format, but be careful; you might get some unexpected results! We're specifically working with millions, but you can apply this technique to any scale.

	A	B	C	D	E	F
1	Region	Smith	Jones	Michaels	Hancock	
2	North	$17,993,906.00	$1,941,972.00	$2,949,318.00	$1,379,624.00	
3	South	$141,897.00	$3,901,525.00	$3,031,663.00	$171,178.00	
4	East	$7,692,284.00	$763,317.00	$1,366,237.00	$182,908.00	
5	West	$2,643,272.00	$1,089,000.00	$1,559,500.00	$21,253.00	
6	Total	$28,471,359.00	$7,695,814.00	$8,906,718.00	$1,754,963.00	
7						

Figure A: Values with lots of digits can be difficult to read.

Let's try a custom format and see what happens:

1. Select the data range and click the Number group's launcher (on the Home tab) or press Ctrl+1 to display the Format Cells dialog.

2. From the Category list (on the Number tab), choose Custom.

3. In the Type control, enter the *$#,," M";* format string (**Figure B**).
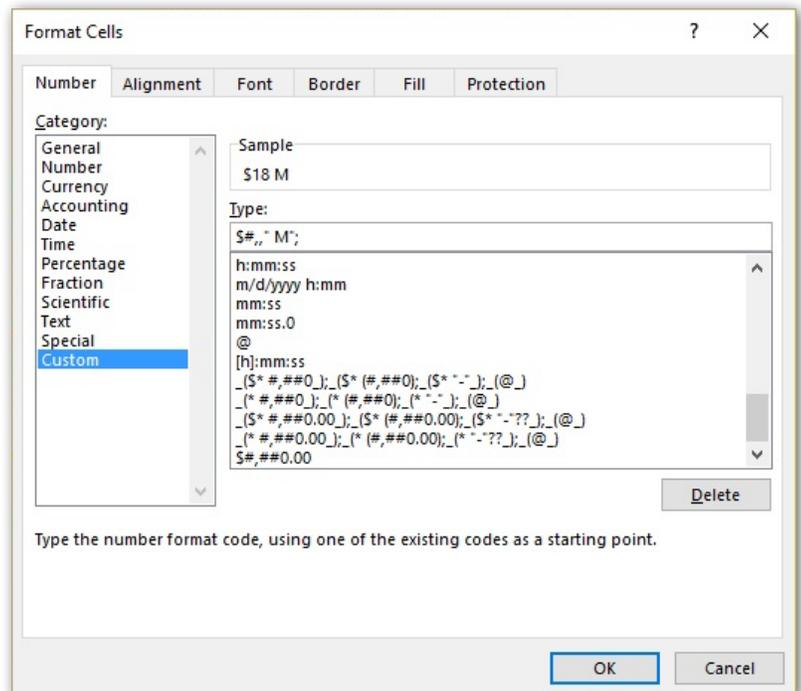

Figure B: Enter a custom format.

4. Click OK to see the formatted results shown in **Figure C**.

	A	B	C	D	E	F
1	Region	Smith	Jones	Michaels	Hancock	
2	North	$18 M	$2 M	$3 M	$1 M	
3	South	$ M	$4 M	$3 M	$ M	
4	East	$8 M	$1 M	$1 M	$ M	
5	West	$3 M	$1 M	$2 M	$ M	
6	Total	$28 M	$8 M	$9 M	$2 M	
7						
8						

Figure C: The formatted values are much easier to read than their full-digit counterparts.

A number format can have up to four sections of code, separated by semicolons. These sections define the format for positive and negative numbers, zero values, and text using the following syntax:

<positive>;<negative>;<zero>;<text>

In this case, the pound sign combined with the two comma characters displays a character in the millions position, if one exists. The *M* component displays a literal M character, to denote millions. There's a space before the M to separate it from the value. You can omit the space if you prefer.

Depending on your needs, this simple format code might not be adequate; you might have noticed that there's no code for negative values. In fact, if you apply this custom format to a negative value, the format displays nothing. To accommodate negative values, use a code similar to one of the following:

$#,," M";($#,," M");

$#,," M";-$#,," M";

$#,," M";[Red]$#,," M";

The code following the first semicolon character applies to negative values:

*   The first code encloses a negative value in parentheses.

*   The second code displays a negative sign before the number.

*   The third displays a negative value in red.

If the format is applied before data entry, users must enter the full value for this format to work as expected. In addition, numbers in the billions and beyond will be displayed in the millions. For instance, the value $1,000,000,000 displays as $1000 M. Value below one million aren't displayed at all; $100,000 becomes $ M.

For a comprehensive list of code symbols, see Create or delete a custom number format.

How to copy a sheet from one workbook to another

Copying a sheet of data from one workbook to another sounds like a complicated job. In truth, Excel has a built-in feature that makes quick work of this task—but as usual, there's more than one way to get the job done. First, I'll show you Excel's built-in route. Then, I'll show you a second method, which you might find a bit easier.

The built-in technique

To copy a sheet from one workbook to another using Excel's user interface, do the following:

1. Open the source and the target workbook; both workbooks must be open for this method to work. The source workbook contains the sheet you want to copy or move. You'll move or copy the source sheet to the target workbook.

2. Right-click the sheet tab in the *source* workbook and choose Move Or Copy from the resulting shortcut menu (**Figure A**).

Figure A: Select Move Or Copy from the shortcut menu.

3. In the Move Or Copy dialog, choose the target sheet from the To Book dropdown (**Figure B**). Excel will display only the open workbooks in this list.

4. If necessary, update the Before Sheet selection.

5. If you want to copy rather than move the sheet, click the Create A Copy option at the bottom of the dialog.

6. Click OK, and Excel will move or copy the sheet to the *target* workbook.

An alternate technique

Now, let's try a second technique. It's not a shortcut or superior to the first method, but you might find it a little more efficient. With both the source and target workbooks open, do the following:

1. Click the View tab and choose View Side By Side in the Window group. Excel will split the screen, horizontally, between the two workbooks.

2. Drag and drop the sheet tab in the source workbook to the target workbook to move the source sheet. To copy the sheet instead of moving it, hold down the Ctrl key while you drag the sheet to the target workbook.
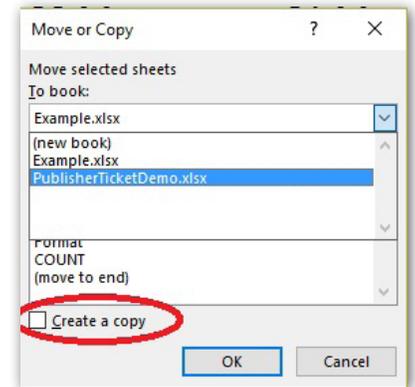
That's all there is to it!

Figure B: Identify the target workbook.

Worth noting

Excel won't overwrite a sheet named the same in the target workbook. Instead, it will rename it by adding a consecutive value to the name.

Moving a sheet isn't without its problems—problems that might show up later. If either workbook contains an expression that references the moved or copied sheet, Excel will do its best to find it, returning an error when it can't. Similarly, removing a sheet referenced by VBA code will result in a runtime error.